MACHINE CUT TASK IDENTIFICATION FOR EFFICIENT PARTITION AND DISTRIBUTION

DESCRIPTION

RELATED APPLICATION

Sub A1

The present application is related to U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0293-US1) entitled "INDEPENDENT NET TASK IDENTIFICATION FOR EFFICIENT PARTITION AND DISTRIBUTION" to Kimelman et al.; U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0465-US1) entitled "NET ZEROING FOR EFFICIENT PARTITION AND DISTRIBUTION" to Roth et al.; and U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0466-US1) entitled "DOMINANT EDGE IDENTIFICATION FOR EFFICIENT PARTITION AND DISTRIBUTION" to Wegman et al. all filed coincident herewith and assigned to the assignee of the present invention.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention generally relates to distributed processing and more particularly, the present invention relates to efficiently assigning tasks across multiple computers for distributed processing.

Background Description

Any large, multifaceted project, such as a complex computer program, may be segmented into multiple smaller manageable tasks. The tasks then may be distributed amongst a group of individuals for independent completion, e.g., an engineering design project, distributed processing or, the layout of a complex electrical circuit such as a microprocessor. Ideally, the tasks are matched with the skills of the assigned individual and each task is completed with the same effort level as every other task. However, with such an ideal matched task assignment, intertask communication can become a bottleneck to project execution and completion. Thus, to minimize this potential bottleneck, it is important to cluster together individual tasks having the highest level of communication with each other. So, for example, in distributing eight equivalent tasks to pairs of individuals at four locations, (e.g., eight design engineers in four rooms) optimally, pairs of objects or tasks with the highest communication rate with each other are assigned to individual pairs at each of the four locations.

Many state of the art computer applications are, by nature, distributed applications. End-users sit at desktop workstations or employ palmtop information appliances on the run, while the data they need to access resides on distant data servers, perhaps separated from these end-users by a number of network tiers. Transaction processing applications manipulate data spread across multiple servers. Scheduling applications are run on a number of machines that are spread across the companies of a supply chain, etc.

When a large computer program is partitioned or segmented into modular components and the segmented components are distributed over two or more machines, for the above mentioned reasons, component placement can have a significant impact on

program performance. Therefore, efficiently managing distributed programs is a major challenge, especially when components are distributed over a network of remotely connected computers. Further, existing distributed processing management software is based on the assumption that the program installer can best decide how to partition the program and where to assign various program components. However, experience has shown that programmers often do a poor job of partitioning and component assignment.

So, a fundamental problem facing distributed application developers is application partitioning and component or object placement. Since communication cost may be the dominant factor constraining the performance of a distributed program, minimizing inter-system communication is one segmentation and placement objective. Especially when placement involves three or more machines, prior art placement solutions can quickly become unusable, i.e., what is known as NP-hard. Consequently, for technologies such as large application frameworks and code generators that are prevalent in object-oriented programming, programmers currently have little hope of determining effective object placement without some form of automated assistance. En masse inheritance from towering class hierarchies, and generation of expansive object structures leaves programmers with little chance of success in deciding on effective partitioning. This is particularly true since current placement decisions are based solely on the classes that are written to specialize the framework or to augment the generated application.

Furthermore, factors such as fine object granularity, the dynamic nature of object-based systems, object caching, object replication, ubiquitous availability of surrogate system objects on every machine, the use of factory and command patterns, etc., all make partitioning in an object-oriented domain even more difficult. In particular, for conventional graph-based approaches to partitioning distributed applications, fine-grained

object structuring leads to enormous graphs that may render these partitioning approaches impractical.

Finally, although there has been significant progress in developing middleware and in providing mechanisms that permit objects to inter-operate across language and machine boundaries, there continues to be little to help programmers decide object-system placement. Using state of the art management systems, it is relatively straightforward for objects on one machine to invoke methods on objects on another machine as part of a distributed application. However, these state of the art systems provide no help in determining which objects should be placed on which machine in order to achieve acceptable performance. Consequently, the initial performance of distributed object applications often is terribly disappointing. Improving on this initial placement performance is a difficult and time-consuming task.

Accordingly, there is a need for a way of automatically determining the optimal program segmentation and placement of distributed processing components to minimize communication between participating distributed processing machines.

SUMMARY OF THE INVENTION

It is therefore a purpose of the present invention to improve distributed processing performance;

It is another purpose of the present invention to minimize communication between distributed processing machines;

It is yet another purpose of the invention to improve object placement in distributed processing applications;

It is yet another purpose of the invention to determine automatically how objects should best be distributed in distributed processing applications

it is yet another purpose of the invention to minimize communication between objects distributed amongst multiple computers in distributed processing applications.

The present invention is a task management system, method and computer program product for determining optimal placement of task components on multiple machines for task execution, particularly for placing program components on multiple computers for distributed processing. First, a communication graph is generated representative of the computer program with each program unit (e.g., an object) represented as a node in the graph. Nodes are connected to other nodes by edges representative of communication between connected nodes. A weight is applied to each edge, the weight being a measure of the level of communication between the connected edges. Terminal nodes representative of the multiple computers are attached to the communication graph. Independent nets may be separated out of the communication graph. A cut is made at each terminal node and the weights of the cut edges are summed. The second heaviest terminal is identified from the cut and edges connected to at least one internal node and not connected to the second heaviest edge are compared against the weight of the second heaviest edge. Any edge found in the comparison to be at least as heavy as the second heaviest terminal node need not be included in the min cut for the communication graph and so, is removed from consideration for the final min cut solution. Finally, program components which may be a single program unit or an aggregate of units are placed on computers according to the communication graph min cut solution.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed preferred embodiment description with reference to the drawings, in which:

Figure 1 shows an example of a flow diagram of the preferred embodiment of the present invention wherein a program is segmented, initially, and initial segments are distributed to and executed on multiple computers;

Figures 2A-C show an example of a communication graph;

Figure 3 is a flow diagram of the optimization steps for determining an optimum distribution of program components;

Figure 4 shows an example of a simple communication graph reducible by the preferred embodiment Machine Cut method of the present invention;

Figure 5 is an example of the Machine Cut method steps of identifying non-terminal edges that may be removed from consideration;

Figure 6 is an example of the steps in contracting or collapsing edges that are at least as heavy as the second heaviest edge.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

As referred to herein, a communication graph is a graphical representation of a multifaceted task such as a computer program. Each facet of the task is an independent task or object that is represented as a node and communication between tasks or nodes is represented by a line (referred to as an edge) between respective communicating nodes. Participating individuals (individuals receiving and executing distributed tasks) are referred to as terminal nodes or machine nodes. A net is a collection of nodes connected

together by edges. Two nets are independent if none of the non-terminal nodes of one net shares an edge with a non-terminal node of the other. Thus, for example, a communication graph of a computer program might include a node for each program object and edges would be between communicating objects, with edges not being included between objects not communicating with each other. In addition, a weight indicative of the level of communication between the nodes may be assigned to each edge. Graphical representation and modeling of computer programs is well known in the art.

Referring now to the drawings, and more particularly, Figure 1 is an example of a flow diagram 100 of the preferred embodiment of the present invention wherein a program is segmented, initially, and initial segments are distributed to and executed on multiple computers. First, in step 102 the communication patterns of a program are analyzed to form a communication graph. Then, in step 104, the traces of the communication graph are analyzed, and an initial partition is determined. In step 106, the partition is optimized for minimum interpartition communication. In step 108, the individual objects are distributed amongst participants for execution according to the optimize partition of step 106.

A component refers to an independent unit of a running program that may be assigned to any participating individual, e.g., computers executing objects of a distributed program. Thus, a component may refer to an instance of an object in an object oriented program, a unit of a program such as Java Bean or Enterprise Java Bean or, a larger collection of program units or components that may be clustered together intentionally and placed on a single participating machine. Further, a program is segmented or partitioned into segments that each may be a single component or a collection of components. After segmenting, analyzing the segments and assigning each of segments

or components to one of the multiple participating machines or computers according to the present invention, the final machine assignment is the optimal assignment.

Thus, a typical communication graph includes multiple nodes representative of components with weighted edges between communicating nodes. Determining communication between components during program execution may be done using a typical well known tool available for such determination. Appropriate communication determination tools include, for example, Jinsight, for Javå applications that run on a single JVM, the Object Level Tracing (OLT) tool, for WebSphere applications or, the monitoring tool in Visual Age Generator.

Figures 2A-C show an example of a communication graph of a net 110 that includes multiple nodes 112, 114, 116, 118, 120 and 122. Each node 112, 114, 116, 118, 120 and 122 represents a program component connected to communication edges 124, 126, 128, 130, 132, 134, 136 and 138 to form the net 110. Adjacent nodes are nodes that share a common edge, e.g., nodes 114 and 122 share edge 126. Each edge 124, 126, 128, 130, 132, 134, 136 and 138 has been assigned a weight proportional to, for example, the number of messages between the adjacent components.

In Figure 2B, Machine nodes 140, 142 and 144 representative of each participating machine (three in this example) are shown connected by edges 146, 148, 150. Initially, a node 112, 114, 116, 118, 120 and 122 may be placed on a machine 140, 142, 144 by adding an edge 146, 148, 150 with infinite weight (indicating constant communication) between the node and the machine. Typically, initial assignment places nodes with specific functions (e.g., database management) on a machine suited for that function. After the initial placement assigning some nodes 112, 114 and 122 to machines 140, 142, 144, other nodes 116, 118, 120 are assigned to machines 140, 142, 144, if they

communicate heavily with a node 112, 114, 122 already assigned to that machine 140, 142, 144. Additional assignment is effected by selectively collapsing edges, combining the nodes on either end of the collapsed edge and re-assigning edges that were attached to one of the two former adjacent nodes to the combined node. When assignment is complete, all of the nodes 112, 114, 116, 118, 120 and 122 will have been placed on one of the machines at terminal nodes 140, 142, 144 and the final communication graph may be represented as terminal nodes 140, 142, 144 connected together by communication edges.

For this subsequent assignment, the graph is segmented by cutting edges and assigning nodes to machines as represented by 152, 154 and 156 in Figure 2C to achieve what is known in the art as a minimal cut set or min cut set. A cut set is a set of edges that, if removed, eliminate every path between a pair of terminal nodes (machine nodes) in the graph. A min cut set is a cut set wherein the sum of the weights of the cut set edges is minimum. While there may be more than one min cut set, the sum is identical for all min cut sets. A min cut may be represented as a line intersecting the edges of a min cut set. So, in the example of Fig. 2C, the sum of the weights of edges 124, 126, 128, 132 and 138 is 2090, which is cost of the cut and is representative of the total number of messages that would be sent between machines at terminal nodes 140, 142, 144 with this particular placement. The min cut identifies the optimum component placement with respect to component communication. While selecting a min cut set may be relatively easy for this simple example, it is known to increase in difficulty exponentially with the number of nodes in the graph.

Figure 3 is a flow diagram 160 of the optimization steps for determining an optimum distribution of program components to individual participating computers according to a preferred embodiment of the present invention. First, in step 162, an initial

communication graph is generated for the program. Then, in step 164 machine nodes are added to the communication graph. As noted above, certain types of components are designated, naturally, for specific host machine types, e.g., graphics components are designated for clients with graphics capability or, server components designated for a data base server. After assigning these host specific components, in step 168 independent nets are identified and the communication graph is partitioned into the identified independent nets as described in U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0293-US1) entitled "INDEPENDENT NET TASK IDENTIFICATION FOR EFFICIENT PARTITION AND DISTRIBUTION" to Kimelman et al. assigned to the assignee of the present invention and incorporated herein by reference. In step 170 the Machine Cut reduction method described hereinbelow is used to reduce the independent nets and then, in step 172 a min cut for the reduced independent nets, the min cuts for all of the independent nets being the min cut for the whole communication graph.

Figure 4 example of a simple communication graph 180 reducible by the preferred embodiment Machine Cut method of the present invention. In this example, the graph 180 includes five (5) non-terminal nodes 182, 184, 186, 188 and 190 connected together by edges 192, 194, 196, 198, 200 and 202, referred to herein as non-terminal edges. Three (3) terminal nodes 204, 206 and 208 are connected to respective non-terminal nodes 182, 184, 186, 188 and 190 by edges 210, 212, 214, 216 and 218, referred to herein as non-terminal edges. A weight is represented as being attached to each edge 192 - 202 and 210 - 218. Dotted line 220 represents a terminal cut at terminal node 204 cutting terminal edges 210, 212. Dotted line 222 represents a terminal cut at terminal node 206 cutting terminal edges 214, 216. Dotted line 224 represents a terminal cut at terminal node 208 cutting terminal edge 218. Essentially, the Machine Cut method eliminates from inclusion in the min cut solution, any terminal or non-terminal edge with heavier communication (i.e., its weight exceeds) than all but the terminal node with the heaviest

level of communication. Thus, in this example, edge 202 is heavier than terminal cut 222. So, edge 202 can be excluded from consideration for inclusion in the min cut solution. Preferably, edge 202 is collapsed, combining nodes 182 and 188, as well as merging (then) parallel edges 198 and 200.

Figure 5 is an example of the Machine Cut method steps 230 of identifying non-terminal edges that may be removed from consideration according to the preferred embodiment of the present invention. First, in step 232, an independent net is selected for reduction. In step 234 terminal cuts are made at each terminal node on the selected net. For each terminal cut, the weights of the edges at the terminal are summed, the sum being the terminal's weight. Then, in step 236, the second heaviest terminal node (the terminal with the second heaviest weight) is identified. In step 238, edges are checked to determine if they are at least as heavy as the identified second heaviest cut weight. All edges connected to at least one non-terminal node are checked in step 238, except that those edges connected to the second heaviest node are excluded. If no edges are found that are as heavy or heavier than the second heaviest cut weight, in step 240, it is determined that the Machine Cut method is unable to reduce the net and in step 242, net reduction ends. Otherwise, in step 244, each edge that was identified in step 238 need not be part of the min cut solution and so, is collapsed. In step 246 it is determined that the independent net has been reduced using the Machine Cut method 230 and, net reduction ends in step 242.

In other words, for each terminal node 204, 206, 208 the weight of all connected edges are summed. Then, the summed are sorted in descending order and the second largest weight is selected and labeled W_{2nd}, for example. Next, any edge 192 - 202, 210, 212 and 218 not connected to the second heaviest node but connected to at least one non-terminal node are compared against W_{2nd}. Any compared edge that is at least as

heavy as W_{2nd} need not be part of the (only) multiway minimum cut of the graph, and as such, may be collapsed. Collapsing each edge results in a simpler graph wherein the min cut solution can be found much more quickly and efficiently, with the min cut solution weight being the same as the original unreduced graph.

Figure 6 is an example of the steps in contracting or collapsing edges that are at least as heavy as the second heaviest edge 244. First in step 2440 the two nodes connected by the collapsed edge are merged, resulting in a single merged node that includes the components of both original nodes. Then, in step 2442 the collapsed edge is discarded. Finally, in step 2444 any "parallel" edge groups (edges connecting the merged node to the same adjacent node) resulting from the merger are replaced with a single edge with its weight equal to the sum of parallel edge weights. Thus, as a result of contracting dominant edges, the graph has been reduced wherein a min cut solution may be found with less effort.

In the preferred embodiment, the min cut step 170 is an iterative process, wherein independent nets are reduced using the Machine Cut steps described herein and, when necessary, in combination with other linear complexity methods such as the Dominant Edge identification method of U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0466-US1) entitled "DOMINANT EDGE IDENTIFICATION FOR EFFICIENT PARTITION AND DISTRIBUTION" to Wegman et al. and the Net Zeroing method of U.S. Patent Application No. 09/_____ (Attorney Docket No. YOR9-2000-0465-US1) entitled "NET ZEROING FOR EFFICIENT PARTITION AND DISTRIBUTION" to Roth et al., all filed coincident herewith, assigned to the assignee of the present invention and incorporated herein by reference. Further, as independent nets are reduced, those reduced nets are further checked as in step 168 above to determine if they may be divided into simpler independent nets. Then, the Machine Cut method of the

Sub G3

preferred embodiment is applied to those simpler independent nets. To reach a solution more quickly, on each subsequent pass, only nodes and edges of a subgraph that were adjacent to areas reduced previously are rechecked. Thus, the communication graph is simplified by eliminating machine cut edges to reach a min cut solution much quicker and much more efficiently than with prior art methods.

The reduction method of the preferred embodiment reduces the number of independent components in the communication graph of a complex program. In the best case, an appropriate allocation of every component in the program is provided. However, even when best case is not achieved, the preferred embodiment method may be combined with other algorithms and heuristics such as the branch and bound algorithm or the Kernighan-Lin heuristic to significantly enhance program performance. Experimentally, the present invention has been applied to communication graphs of components in several programs with results that show significant program allocation improvement, both in the quality of the final solution obtained and in the speed in reaching the result.

Although the preferred embodiments are described hereinabove with respect to distributed processing, it is intended that the present invention may be applied to any multi-task project without departing from the spirit or scope of the invention. Thus, for example, the task partitioning and distribution method of the present invention may be applied to VLSI design layout and floor planning, network reliability determination, web pages information relationship identification, and "divide and conquer" combinatorial problem solution approaches, e.g., "the Traveling Salesman Problem."

While the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.